TEAS Working Group   K. Lam
Internet Draft   E. Varma
Intended status: Informational   Alcatel-Lucent
Intended status: Informational   P. Doolan
Expires: April 2016   Coriant
                                                              N. Davis
                                                                 Ciena
                                                             B. Zeuner
                                                      Deutsche Telekom
                                                              M. Betts
                                                                   ZTE
                                                               I. Busi
                                                                Huawei
                                                          S. Mansfield
                                                              Ericsson
                                                            R. Vilalta
                                                                  CTTC
                                                              V. Lopez
                                                            Telefonica
                                                      October 16, 2015

Usage of IM for network topology to support TE Topology YANG Module Development
draft-lam-teas-usage-info-model-net-topology-02.txt

Status of this Memo

Abstract

The benefits of using a common Information Model (IM) as a foundation for deriving purpose and protocol specific interfaces, particularly for complex networking domains, has been described in draft-betts-netmod-framework-data-schema-uml. This draft describes an existing information model relevant to Network Topology and illustrates how it can be used to help ensure the consistency and completeness of the YANG data model for TE topologies solutions work in TEAS.

Table of Contents

1. Introduction

This draft describes an existing information model (IM) relevant to Network Topology [ONF TR-512] and illustrates how it can be used to help ensure the consistency and completeness of the YANG data model (DM) for TE topologies solutions development work in TEAS.

2. Background and Motivation

Information Models (IM) and Data Models (DM) are related but different. An IM provides an abstract, conceptual view of the system being modeled in terms of its constituent parts (objects), independent of any specific implementations or protocols used to transport the data; it hides all protocol and implementation details (RFC 3444, TM Forum/NGCOR, ITU-T SG 15). A DM is a concrete specification in a particular language of an interface to, in this case, a controlled/managed system. The intention of the distinction between IMs and DMs has been to separate the modeling of problem space semantics from the modeling of the implementation of those semantics (though the dividing line has not always been clearly articulated).

A DM may be derived from an IM though it is often created without (explicit or obviously implicit) reference to one. When a DM is derived from an IM, the DM and the components of the system it provides control/management access to are traceable to the definitions provided in the IM. There is no ambiguity between designer, developer, user or operator regarding the name, function, and information elements that are associated with a particular managed object.

As described in [I-D.betts], when DMs are created “in isolation” solely for the purpose of encoding specific interfaces, they may do that job adequately for any particular interface but in complex domains may create opportunities for confusion, duplication of effort, lack of interoperability, and lack of extensibility. In the past, ad-hoc development of DMs has caused significant operational and implementation inefficiencies in our industry.

Since March 2014, upon IESG recommendation that SNMP no longer be used for new work re configuration and that NETCONF/YANG be used instead, there has been an explosion of YANG DM development in IETF. It has consequently been recognized as essential to assure proper coordination of YANG DM development (including reaching out to different SDOs/consortia), as well as to assure that the YANG modules themselves provide a good representation of what is being modeled, to meet expectations of functionality, quality, and interoperability. In order to facilitate this objective, guidance from available pertinent IMs can be valuable.

This draft describes an existing information model relevant to Network Topology [ONF TR-512], which is part of the Common Information Model (ONF-CIM) of network resources (as described in [I-D.betts]), that can be leveraged to assess the consistency and completeness of related YANG modules under development. Being part of a Common Information Model, it will not lead to development of incompatible/uncoordinated models that can be difficult to maintain as other purpose-specific interfaces are developed.

3. The Common Information Model

This section provides a high level introduction to the ONF Common Information Model (ONF-CIM), and in particular its Core Model

Fragment (see [ONF TR-512]), to provide an overall context for the topology relevant subset. The ONF-CIM has been developed through collaboration among several SDOs, including ITU-T, TM Forum, and ONF, and also published as ITU-T Recommendation G.7711 [G.7711].

An information model describes the things in a domain in terms of objects, their properties (represented as attributes), and their relationships.

The ONF-CIM is expressed in a formal language called UML (Unified Modeling Language). UML has a number of basic model elements, called UML artifacts. In order to assure a consistent and harmonized modeling approach, only a selected subset of these UML artifacts were used in the development of the ONF-CIM according to guidelines for creating an information model expressed in UML (see the UML Guidelines document in the ONF TR-512 [ONF TR-512]).

The ONF-CIM has been developed using the Papyrus open source UML Tool, for which a detailed guidelines document is available (see the Papyrus Guidelines document in the ONF TR-512 [ONF TR-512]). This guidelines document also describes how the modelers constructing the ONF-CIM can cooperate in the GitHub environment to allow for separate and still coordinated development of the ONF-CIM fragments.

The OMF-CIM includes all of the artifacts (objects, attributes, relationships, etc.) that are necessary to describe the domain for the applications being developed.

It will be necessary to continually expand and refine the ONF-CIM over time as, for example to add, new applications, capabilities or forwarding technologies, or to refine the ONF-CIM as new insights are gained. To allow these extensions to be made in a seamless manner, the ONF-CIM is structured into a number of model fragments. This modeling process allows the fragments that contain these extensions to be developed, by the domain experts, with as much independence as possible. This process is further articulated in [I-D.betts].

3.1. Core Model Fragment

The Core Model Fragment of the ONF-CIM consists of model artifacts that are intended for use by multiple applications and/or forwarding technologies.

For navigability, the Core Model Fragment is further sub-structured into modules. Currently, these consist of a Core Network Module and a Core Foundation Module.

3.1.1. Core Network Module

The Core Network Module (CNM) consists of artifacts that model the essential network aspects that are neutral to the forwarding technology of the network. The CNM currently encompasses Topology, Termination, and Forwarding aspects (subsets of the CNM) as described below:

- Topology Subset of CNM

  The Topology subset of the CNM supports the modeling of network topology information, which can be used to build the topology database and depict the topology. Object classes representing topological entities include:

  o Forwarding Domain (FD): Offers the potential to enable forwarding of information.

  o Link (L): Models the adjacency between two or more FDs. A Link has LinkEnds (LE).

  o Logical Termination Point (LTP): Models the ports of a link. It encapsulates the termination, adaptation, and OAM functions of one or more transport layers.

  o Network Element (NE): While not actually part of topology, a NE brings meaning to the FD and the LTP contexts (and hence the links). A NE represents physical equipment “bundling” to provide a view of management scope, management access, and session.

The Topology subset of the CNM supports network topology abstraction and virtualization. FD abstraction is supported via recursive aggregation and virtualization via partitioning of resources according to the resource dedication criterion.

- Forwarding Subset of CNM

  The Forwarding subset of the CNM (not covered in detail in this draft) supports configuration of forwarding entities, including their setup, modification, and tear down. Artifacts representing the forwarding construct include:

  o ForwardingConstruct (FC): In conjunction with the EndPoint, FC models the enabled forwarding between two EPs across a FD.

  o EndPoint (EP): Models the access to the FC, and associates the FC to the LTP. When the FC supports protection, the EP also indicates its role in the protection scheme, i.e., whether it is a working or protection EP.

  o FcRoute: Also known as SncRoute. It models the individual routes of an FC.

  o FcSwitch: Also known as SncSwitch. It models the switched forwarding of traffic (traffic flow) between EPs and is present where there is protection functionality in the FD.

- Termination Subset of CNM

  The Termination subset of the CNM (not covered in detail in this draft) supports modeling of the processing of transport characteristic information, such as termination, adaptation, OAM, etc. Artifacts representing the termination and adaptation and OAM construct include:

  o Logical Termination Point (LTP): See the LTP description in the Topology Subset

  o Layer Protocol (LP): This identifies the type of signal and is the anchor for transport layer protocol specific definitions, which are modeled as conditional packages, e.g., for OTN, ODUk_TTP_Pac, OCh_TTP_Pac, etc.

3.1.2. Core Foundation Module

To communicate about an entity, it is important to have some way of referring to that entity, i.e., to have some way of referencing it. The Core Foundation module defines the artifacts for referencing entities; i.e.:

- Global Unique ID (GUID):

  An identifier that is globally unique where an identifier is a property of an entity/role with a value that is unique within an identifier space, where the identifier space is itself unique, and immutable. The identifier therefore represents the identity of the entity/role. An identifier carries no semantics with respect to the purpose of the entity.)

- Local ID:

  An identifier that is unique in the context of some scope that is less than the global scope (where an identifier is as defined in GUID above).

- Name:

  A property of an entity with a value that is unique in some namespace but may change during the life of the entity. A name carries no semantics with respect to the purpose of the entity.

- Label:

  A property of an entity with a value that is not expected to be unique and is allowed to change. A label carries no semantics with respect to the purpose of the entity and has no effect on the entity behavior or state.

The Core Foundation module also provides the opportunity to extend any entity using the Extension structure.

The module also defines two foundation object classes:

- GlobalClass:

  Super class of object classes for which their instances can exist on their own right, e.g. NE, LTP, FD, Link, and FC. Global classes shall have one and only one globally unique identifier (GUID) and may have zero or more local identifiers, zero or more names, zero or more labels, zero or more extensions.

- LocalClass:

  Super class of object classes for which the existence of their instances depends on instances of global classes; e.g., LP (of LTP), EP (of FC), and LE (of Link). Local classes shall have at least one local identifier, may have zero or more names, zero or more labels, zero or more extensions.

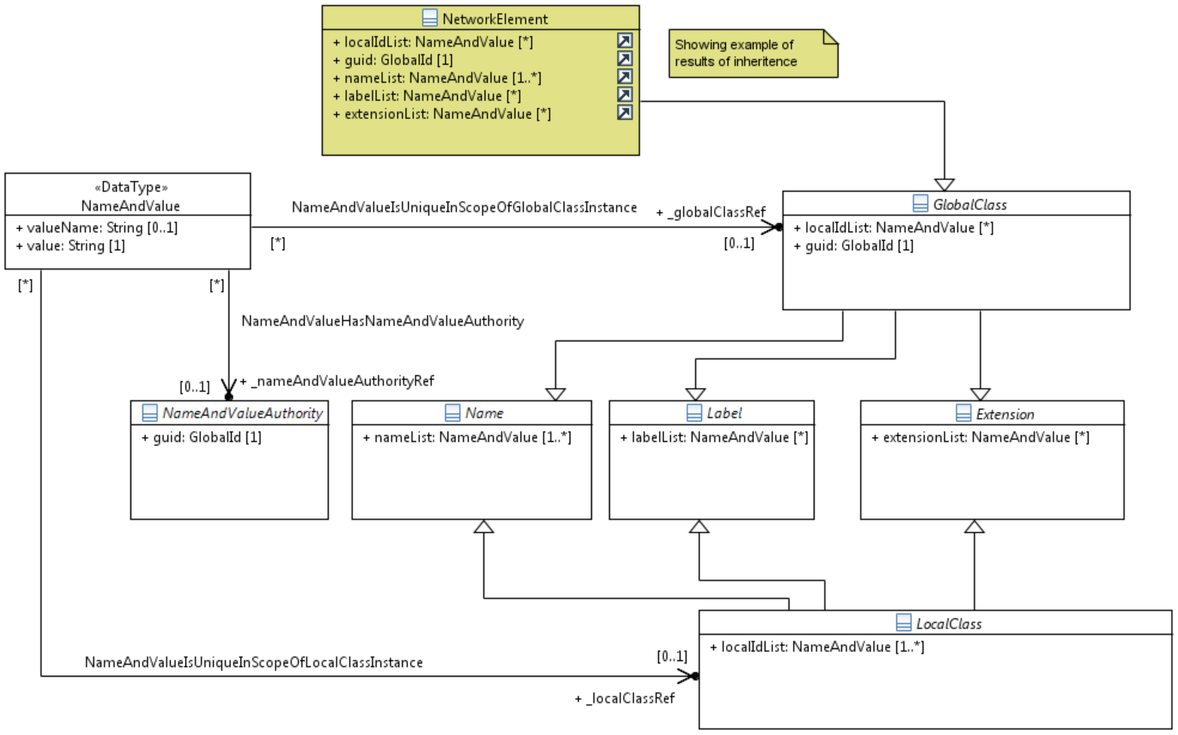
NetworkElement
+ localIdList: NameAndValue [*]
+ guid: GlobalId [1]
+ nameList: NameAndValue [1..*]
+ labelList: NameAndValue [*]
+ extensionList: NameAndValue [*]
Showing example of results of inheritence
«DataType» NameAndValue
+ valueName: String [0..1]
+ value: String [1]
NameAndValueIsUniqueInScopeOfGlobalClassInstance
+ _globalClassRef
[*]
[0..1]
GlobalClass
+ localIdList: NameAndValue [*]
+ guid: GlobalId [1]
[*]
[*]
NameAndValueHasNameAndValueAuthority
[0..1]
+ _nameAndValueAuthorityRef
NameAndValueAuthority
+ guid: GlobalId [1]
Name
+ nameList: NameAndValue [1..*]
Label
+ labelList: NameAndValue [*]
Extension
+ extensionList: NameAndValue [*]
LocalClass
+ localIdList: NameAndValue [1..*]
NameAndValueIsUniqueInScopeOfLocalClassInstance
[0..1]
+ _localClassRef

Figure 3-1 Artifacts for Referencing of Entities

The Core Foundation module also defines a State_Pac artifact, which is a package of state attributes. The State_Pac is inherited by GlobalClass and LocalClass object classes. The State_Pac consists of the following state-related attributes:

- Operational State:

  Read-only with values: DISABLED, ENABLED

- Administrative State:

  Read-only with values: LOCKED, UNLOCKED

- Usage State:

  Read-only with values: IDLE, ACTIVE, BUSY

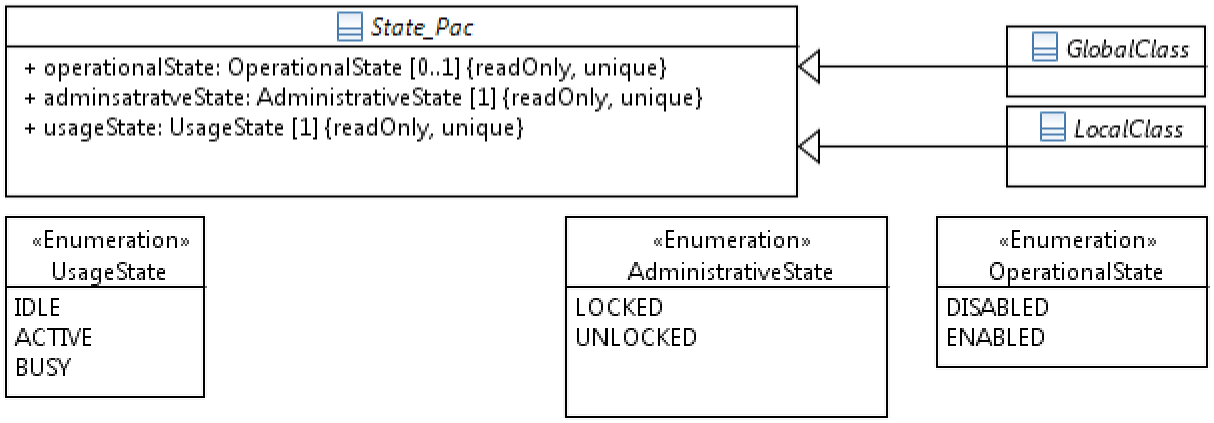
State_Pac
+ operationalState: OperationalState [0..1] {readOnly, unique}
+ adminsatratveState: AdministrativeState [1] {readOnly, unique}
+ usageState: UsageState [1] {readOnly, unique}
GlobalClass
LocalClass
«Enumeration» UsageState
IDLE
ACTIVE
BUSY
«Enumeration» AdministrativeState
LOCKED
UNLOCKED
«Enumeration» OperationalState
DISABLED
ENABLED

Figure 3-2 States of Objects

3.2. Other Fragments

In addition to the Core Fragment, the ONF-CIM contains forwarding technology and application specific fragments. The Optical Transport Fragment of the ONF-CIM (see [ONF TR-512]) encompasses transport technology layers 0, 1, and 2.

4. High Level Description of the Topology Subset of the CNM

This section provides a high-level overview of the Topology Subset of the CNM. Figure 4-1 below is a skeleton class diagram illustrating the key object classes. To avoid cluttering the figure, not all associations have been shown and all of the attributes were omitted.

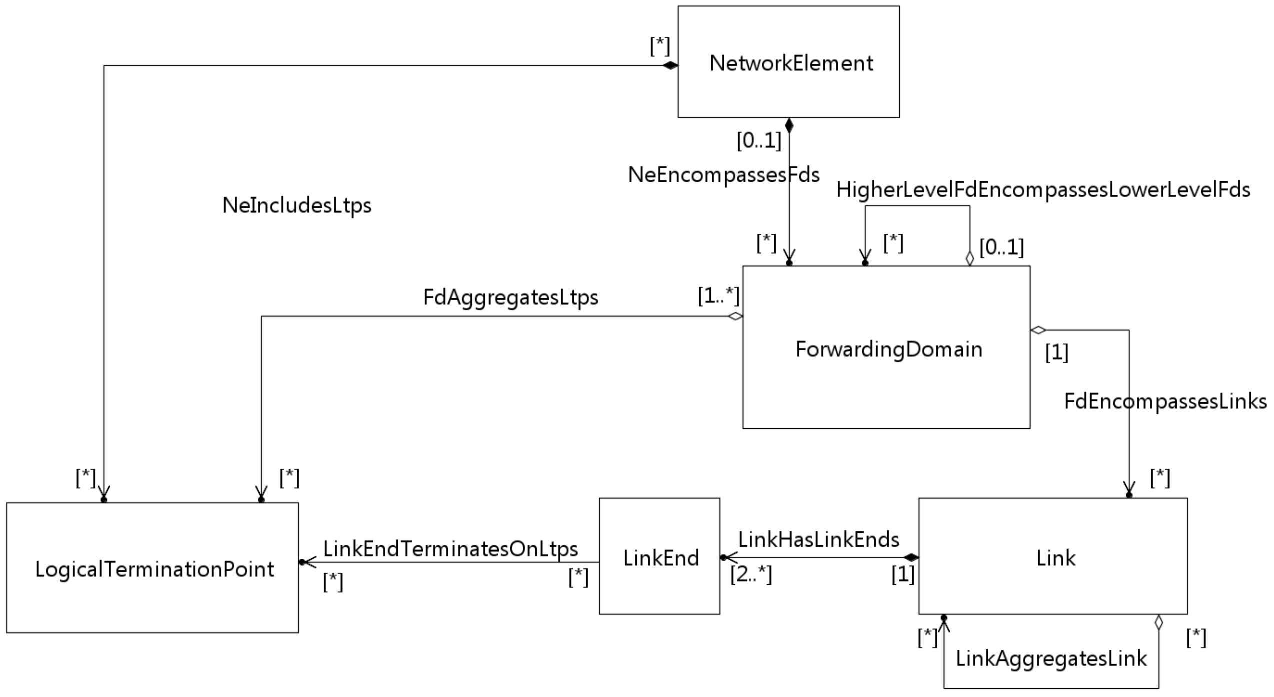

Figure 4-1 Overview of the CNM Topology Subset

4.1. Object Classes of the CNM Topology Subset

This section describes the object classes of the Topology Subset of the CNM. Relationships between these classes are described in section 4.2 below

4.1.1. LogicalTerminationPoint (LTP) and LayerProtocol (LP)

The LogicalTerminationPoint (LTP) object class encapsulates the termination, adaptation and OAM functions of one or more transport protocol layers. The structure of the LTP supports all transport protocols including circuit and packet forms. Each transport layer is represented by a LayerProtocol (LP) instance. The LayerProtocol instances of the LTP can be used for controlling the termination and OAM functionality of that layer. It can also be used for controlling the adaptation (i.e. encapsulation and/or multiplexing of client signal). Where the client/server relationship is fixed 1:1 and immutable, the different layers can be encapsulated in a single LTP instance. Where there is a n:1 relationship between client and server, the layers must be split over separate instances of LTP.

The LP object class is defined with generic attributes "layerProtocolName" for indicating the supported transport layer protocol.

Transport layer specific properties (such as layer-specific termination and adaptation properties) are modeled as attributes of conditional packages (called "_Pacs" in the UML notation of the ONF-CIM) associated with the LP object class.

4.1.2. ForwardingDomain (FD)

The ForwardingDomain (FD) object class models the switching and routing capabilities (see "subnetwork" topological component in [G.852.2] and [TMF612]), which is used to effect forwarding of transport characteristic information and offers the potential to enable forwarding. It represents the resource that supports flows across the FD. The FD object can hold zero or more instances of ForwardingConstruct (FC) (representing constrained forwarding, not discussed further in this document, covering connections, VLANs etc) of one or more layer networks; e.g., OCh, ODU, ETH, and MPLS-TP. The FD object provides the context for operations that create/modify/delete FCs.

The FD object class supports a recursive aggregation relationship such that the internal construction of an FD can be exposed as multiple lower level FDs and associated Links (partitioning) (see section 4.2.1.)

At the lowest level of recursion, a FD (within a network element) could represent a switch matrix (i.e., a fabric).

Note that an NE can encompass multiple switch matrices (FDs), as described in section 4.2.2. An instance of FD is associated with zero or more LTP objects, as described in section 4.2.3.

4.1.3. Link and Link End (LE)

The Link object class models the adjacency between two or more ForwardingDomains (FDs).

In its basic form (i.e., point-to-point Link) it associates a set of LTP clients on one FD with an equivalent set of LTP clients on another FD. Like the FC, the Link has endpoints (LinkEnd) which take roles in the context of the function of the Link. A point-to-point Link can be a TE Link and support parameters such as capacity, delay etc. These parameters depend on the type of technology that supports the link.

A Link can be terminated on two or more FDs. This provides support for technologies such as PON and Layer 2 MAC in MAC configurations.

The LinkEnd further details the relationship between FD and Link for asymmetric cases.

A FD may aggregate Links (see section 4.2.5).

The Link can support multiple transport layers via the associated LTP object. An instance of Link can be formed with the necessary properties according to the degree of virtualization. For implementation optimization, multiple layer-specific links can be merged and represented as a single Link instance.

4.1.4. Network Element (NE)

The NetworkElement (NE) object class represents a network element (traditional NE) in the data plane or a virtual network element visible in an interface where virtualization is used.

In the direct interface from a SDN controller to a network element in the data plane, the NE object defines the scope of control for the resources within the network element, e.g., internal transfer of user information between the external terminations (ports), encapsulation, multiplexing/demultiplexing, and OAM functions, etc. The NE provides the scope of the naming space for identifying objects representing the resources within the network element.

Where virtualization is employed, the NE object represents a virtual NE (VNE). The mapping of the VNE to the NEs is the internal matter of the SDN controller that offers the view of the VNE. Via the interface between hierarchical SDN controllers, NE instances can be created (or deleted) for providing (or removing) virtual views of the combination of slices of network elements in the data plane.

4.2. Relationships between Object Classes of the Topology Subset

4.2.1. ForwardingDomain Recursive Aggregation (HigherLevelFdEncompassesLowerLevelFds Aggregation)

Figure 4-2 below provides a pictorial example of ForwardingDomain (FD) recursion with Links.

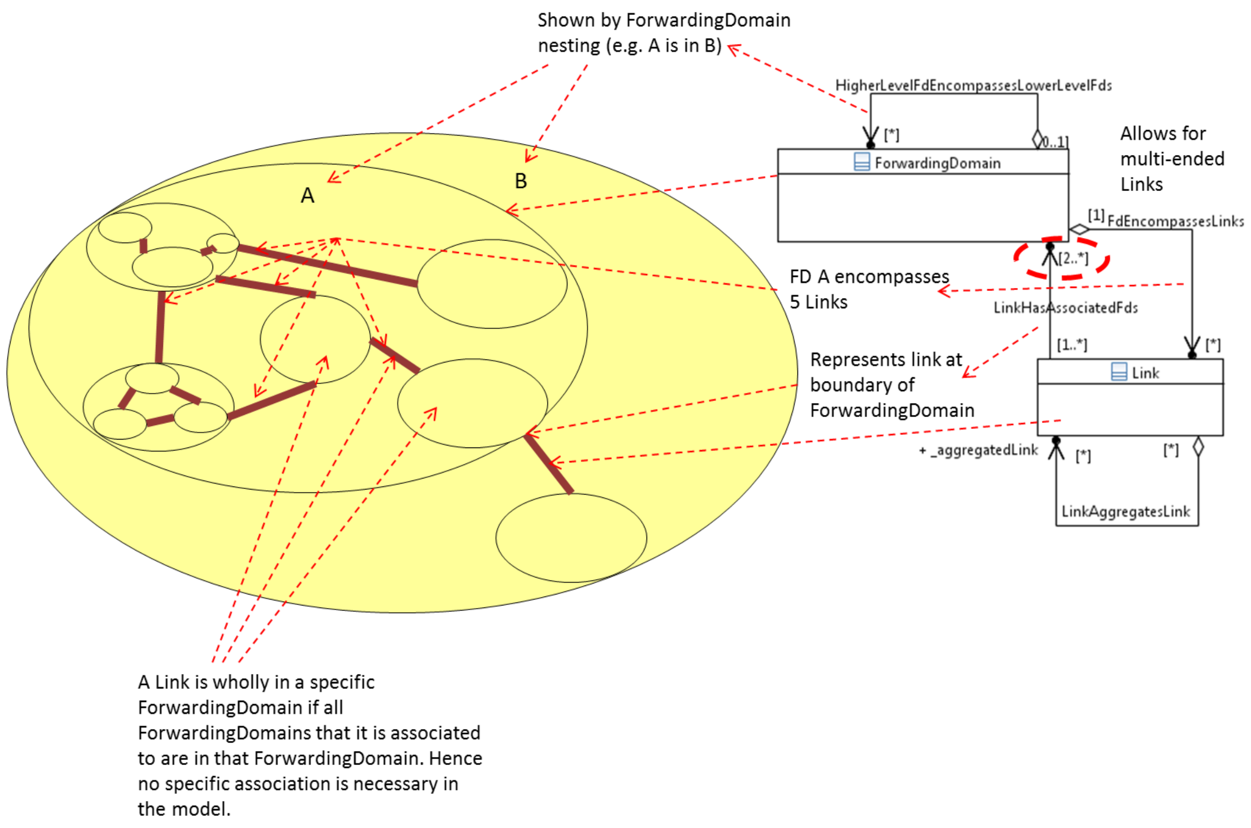
Shown by ForwardingDomain nesting (e.g. A is in B)
HigherLevelFdEncompassesLowerLevelFds
[*]
0..1]
ForwardingDomain
Allows for multi-ended Links
[1] FdEncompassesLinks
A
B
[2..*]
FD A encompasses 5 Links
LinkHasAssociatedFds
[1..*]
[*]
Represents link at boundary of ForwardingDomain
Link
+ _aggregatedLink
[*]
[*]
LinkAggregatesLink
A Link is wholly in a specific ForwardingDomain if all ForwardingDomains that it is associated to are in that ForwardingDomain. Hence no specific association is necessary in the model.

Figure 4-2 ForwardingDomain recursion with Links

Figure 4-2 shows a UML fragment including the Link and ForwardingDomain (FD). For simplicity it is assumed here that the Links and FDs are for a single LayerProtocol (LP) although it can be seen from the detailed figure earlier in this section that both a FD and link can support a list of LPs.

The pictorial form shows a number of instances of FD interconnected by Links and shows nesting of FDs. The recursive aggregation "HigherLevelFdEncompassesLowerLevelFds" relationship (represented by an open diamond) supports the FD nesting but it should be noted that this is intentionally showing no lifecycle dependency between the lower FDs and the higher ones that nest them (to do this composition, a black diamond would have been used instead of the open diamond).

This is to allow for rearrangements of the FD hierarchy (e.g. when regions of a network are split or merged). This emphasizes that the nesting is an abstraction rather than decomposition. The underlying network still operates regardless of how it is perceived in terms of aggregating FDs. The model allows for only one hierarchy.

4.2.2. Network Elements encompassing ForwardingDomains (NeEncompassesFds Aggregation)

Figure 4-3 below provides a pictorial example of ForwardingDomain (FD) recursion with Links and NEs.

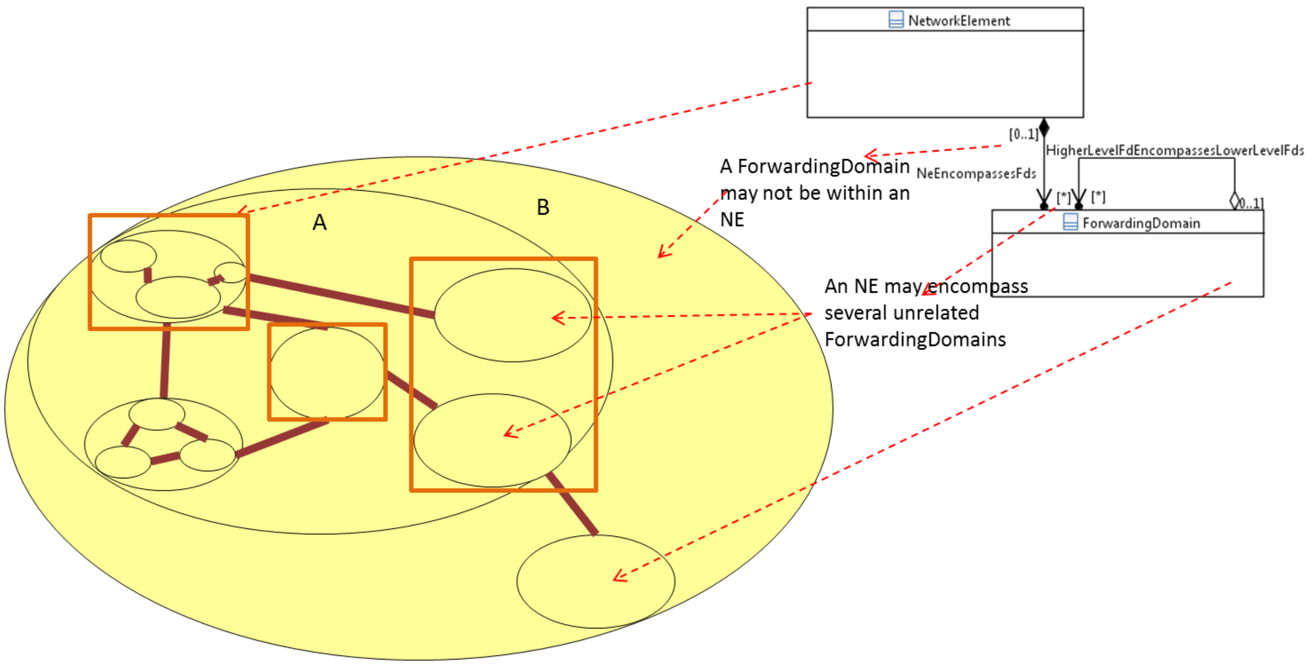
NetworkElement
[0..1]
HigherLevelFdEncompassesLowerLevelFds
A ForwardingDomain may not be within an NE
NeEncompassesFds
[*]
[*]
0..1]
ForwardingDomain
A
B
An NE may encompass several unrelated ForwardingDomains

Figure 4-3 ForwardingDomain recursion with Links and NEs

Figure 4-3 above shows an overlay of NetworkElement (NE) on the ForwardingDomains and a corresponding fragment of UML showing only the ForwardingDomain and NetworkElement classes.

The figure emphasizes that one level of abstraction of ForwardingDomain is bounded by an NE. This is represented in the UML fragment by the composition association (black diamond) that explains that there is a lifecycle dependency in that the ForwardingDomain at this level that cannot exist without the NE. The figure also shows that a ForwardingDomain need not be bounded by an NE (as explained in the UML fragment by the 0..1 composition) and that a ForwardingDomain may have smaller scope than the whole NE (even when considering only a single LayerProtocol as described below).

In one of the cases depicted (e.g., the right hand side NE encompassing two FDs), the two ForwardingDomains in the NE are completely independent. In the other cases depicted (e.g., the left hand side NE encompassing three FDs) the subordinate ForwardingDomains are themselves joined by Links emphasizing that the NE does not necessarily represent the lowest level of relevant network decomposition.

The figure also emphasizes that just because one ForwardingDomain at a particular level of decomposition of the network happens to be the one bounded by an NE does not mean that all ForwardingDomains at that level are also bounded by NEs.

4.2.3. ForwardingDomain association with LTPs (FdAggregatesLtps Composition)

An instance of FD is associated with zero or more LTP objects via the "FdAggregatesLtps" composition.

4.2.4. ForwardingDomain aggregating Links (FdEncompassesLinks)

A ForwardingDomain can aggregate links. An example of ForwardingDomain Recursive Aggregation with Links is shows in section 4.2.1 above.

However, the FdAggregatesLink association is not modeled because this association can be inferred from the higherLevelFdContainsLowerLevelFd association together with the linkHasAssociatedFds association.

4.2.5. ForwardingDomain aggregating NEs

A ForwardingDomain can aggregate Network Elements. An example of ForwardingDomain Recursive Aggregation with Links and NEs is shown in section 4.2.2 above.

However, the FdAggregatesNe association is not modeled because this association can be inferred from higherLevelFdContainsLowerLevelFd association and together with the NeEncompassesFd association.

5. Detailed Description of the Topology Subset

The two key classes related to Topology are the ForwardingDomain (FD) and the Link. For simple cases the FD represents the switching capability in the network and the Link represents adjacency. These are depicted in the context of other model classes in Figure 5-1.

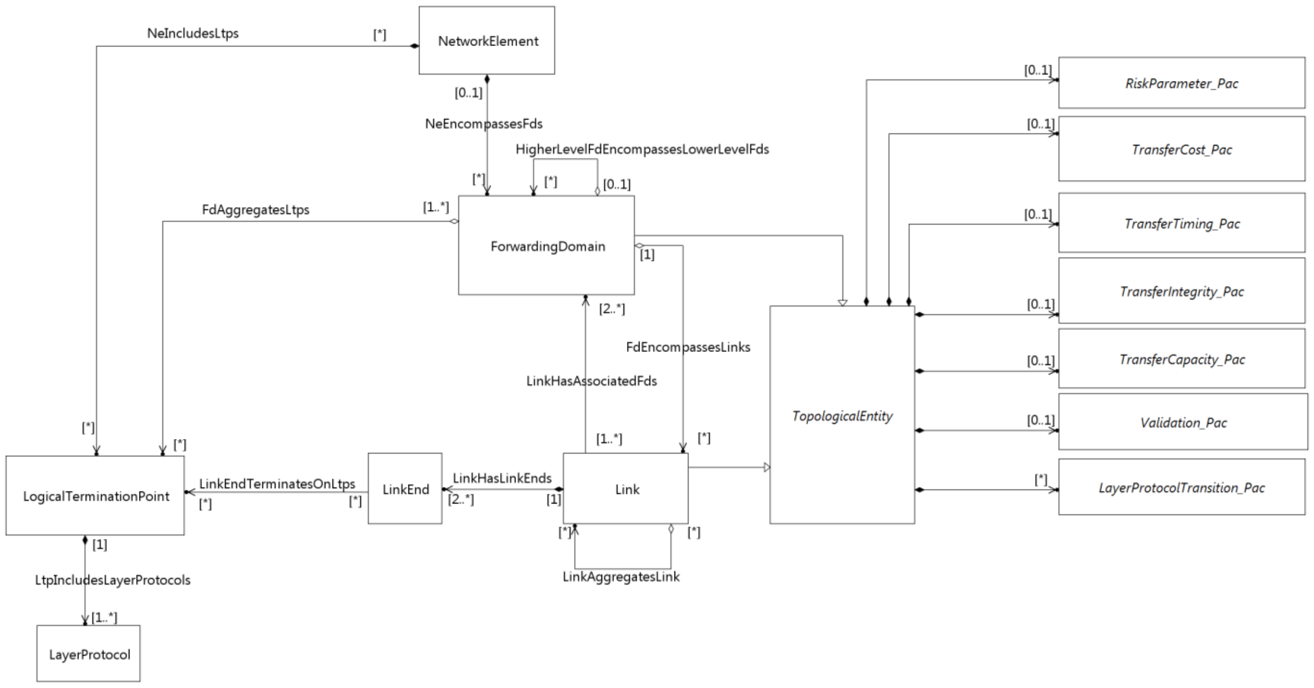

Figure 5-1 Object Classes and Relationships in the Topology Subset

Figure 5-1 shows a lightweight view of the model omitting the attributes (where appropriate these will be described later in this section).

The FD and Link will be described in detail later in the document. Figure 5-1 focuses on interrelationships and these will be the focus of this section. The figure shows that:

- An FD may be a subordinate part of a NetworkElement (NE) or may be larger than, and independent of, any NE.

- An FD may encompass lower level FDs. This may be such that:

  - A FD directly contained in an NE is divided into smaller parts

  - A FD not encompassed by an NE is divided into smaller parts some of which may be encompassed by NEs

  - The FD represents the whole network

- An FD encompasses Links that interconnect any FDs encompassed by the FD

- A Link may aggregate Links in several ways

  - In parallel where several links are considered as one

  - In series where Links chain to form a Link of a greater span

    - Note that this case requires further development in the model

- A Link has associated FDs that it interconnects

  - A Link may interconnect 2 or more FDs

    - Note that it is usual for a Link to interconnect 2 FDs but there are cases where many FDs may be interconnected by a Link

- A Link has LinkEnds (LE)that represent the ports of the Link itself
  - LEs are especially relevant for multi-ended asymmetric Link
- An LE aggregates LogicalTerminationPoints (LTPs) that bound the Link. The LTP represent a stack LayerProtocol terminations where the details of each is held in the LayerProtocol (LP). The LTP may be:
  - Part of an NE
  - Conceptually independent from any NE
- An LE references LTPs on which the Link associated to the LE terminates

Both the Link and FD are TopologicalEntities (an abstract class, i.e. a class that will never instantiate) and hence they can acquire contents from the conditional packages (_Pacs). The conditional packages provide all key topology properties.

5.1. Topological Entity

As noted in the previous section the two key topology classes are Forwarding Domain (FD) and Link (L).

The FD topological component is used to show the potential to enable forwarding. At the lowest level of recursion, an FD (within a network element (NE)) represents a switch matrix (e.g., a fabric). Note that an NE can encompass multiple switch matrices (FDs).

As noted earlier the Link models adjacency between two or more Forwarding Domains (FD).

Both the link and the FD have the potential to handle more than one layerProtocol (both have a layerProtocolNameList attribute).

As shown in Figure 5-1 an object class “TopologicalEntity” has been defined to collect topology-related properties (characteristics etc.) that are common for FD and Link.

A TopologicalEntity is an abstract representation of the emergent effect of the combined functioning of an arrangement of components (running hardware, software running on hardware, etc). The effect can be considered as the realization of the potential for apparent communication adjacency for entities that are bound to the terminations at the boundary of the TopologicalEntity.

The TopologicalEntity enables the creation of constrained forwarding to achieve the apparent adjacency. The apparent adjacency has intended performance degraded from perfect adjacency and a statement of that degradation is conveyed via the attributes of the packages associated with this class. In the model both ForwardingDomain and Link are TopologicalEntities.

This abstract class is used as a modeling approach to apply packages of attributes to both Link and ForwardingDomain. Link and ForwardingDomain are the key TopologicalEntities.

5.2. Characteristics of Topological Entity

As noted above the characteristic of a TopologicalEnity are covered by the conditional packages (_PACs).

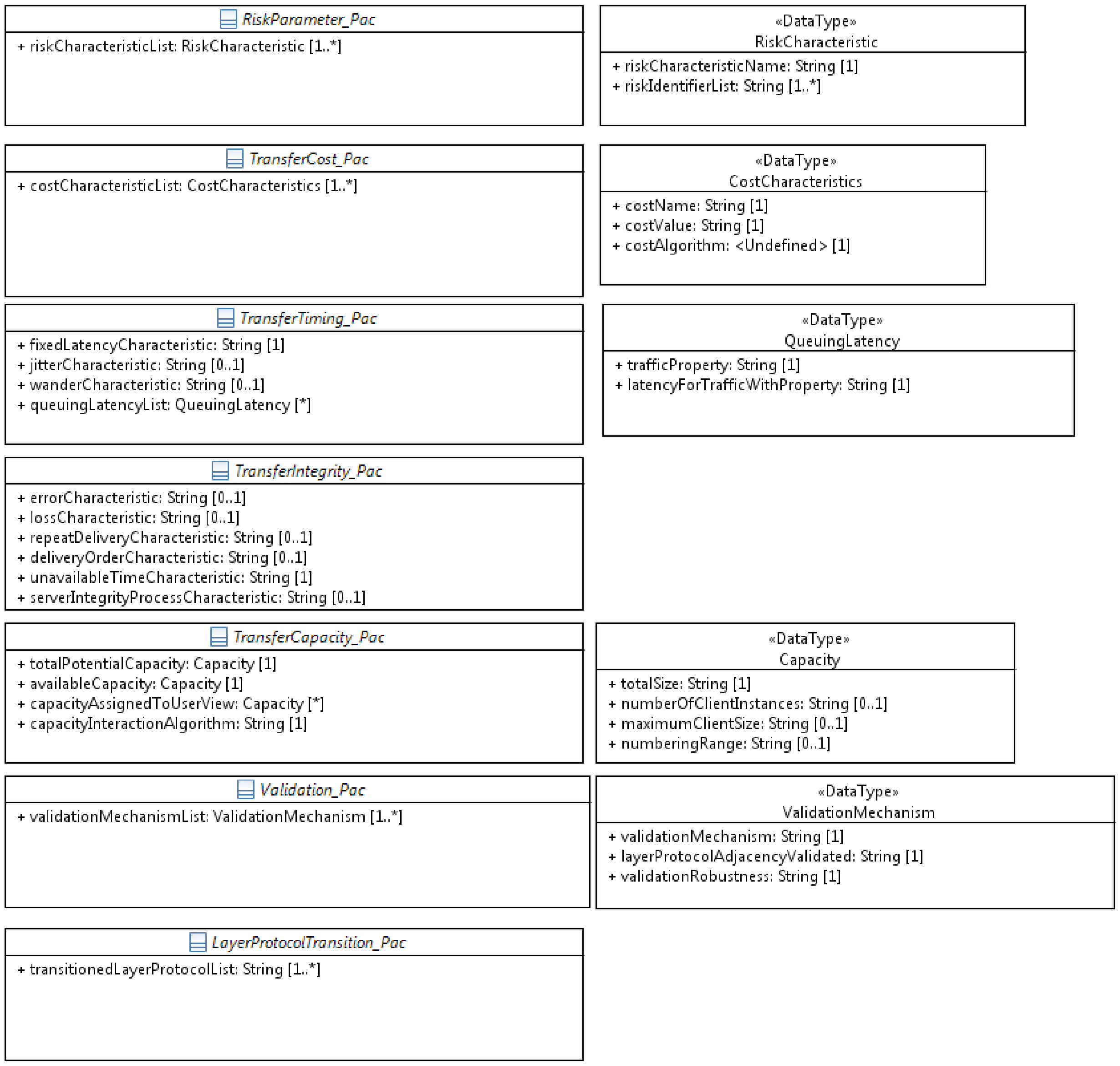
RiskParameter_Pac
+ riskCharacteristicList: RiskCharacteristic [1..*]
«DataType» RiskCharacteristic
+ riskCharacteristicName: String [1]
+ riskIdentifierList: String [1..*]
TransferCost_Pac
+ costCharacteristicList: CostCharacteristics [1..*]
«DataType» CostCharacteristics
+ costName: String [1]
+ costValue: String [1]
+ costAlgorithm: <Undefined> [1]
TransferTiming_Pac
+ fixedLatencyCharacteristic: String [1]
+ jitterCharacteristic: String [0..1]
+ wanderCharacteristic: String [0..1]
+ queuingLatencyList: QueuingLatency [*]
«DataType» QueuingLatency
+ trafficProperty: String [1]
+ latencyForTrafficWithProperty: String [1]
TransferIntegrity_Pac
+ errorCharacteristic: String [0..1]
+ lossCharacteristic: String [0..1]
+ repeatDeliveryCharacteristic: String [0..1]
+ deliveryOrderCharacteristic: String [0..1]
+ unavailableTimeCharacteristic: String [1]
+ serverIntegrityProcessCharacteristic: String [0..1]
TransferCapacity_Pac
+ totalPotentialCapacity: Capacity [1]
+ availableCapacity: Capacity [1]
+ capacityAssignedToUserView: Capacity [*]
+ capacityInteractionAlgorithm: String [1]
«DataType» Capacity
+ totalSize: String [1]
+ numberOfClientInstances: String [0..1]
+ maximumClientSize: String [0..1]
+ numberingRange: String [0..1]
Validation_Pac
+ validationMechanismList: ValidationMechanism [1..*]
«DataType» ValidationMechanism
+ validationMechanism: String [1]
+ layerProtocolAdjacencyValidated: String [1]
+ validationRobustness: String [1]
LayerProtocolTransition_Pac
+ transitionedLayerProtocolList: String [1..*]

Figure 5-2 Conditional Packages of Topological Entity

5.2.1. Risk (RiskParameter_Pac)

The risk characteristics of a TopologicalEntity come directly from the underlying physical realization.

The risk characteristics propagate from the physical realization to the client and from the server layer to the client layer, this propagation may be modified by protection.

A TopologicalEntity may suffer degradation or failure as a result of a problem in a part of the underlying realization.

The realization can be partitioned into segments which have some relevant common failure modes.

There is a risk of failure/degradation of each segment of the underlying realization.

Each segment is a part of a larger physical/geographical unit that behaves as one with respect to failure (i.e. a failure will have a high probability of impacting the whole unit (e.g. all fibers in the same cable).

Disruptions to that larger physical/geographical unit will impact (cause failure/errors to) all TopologicalEntities that use any part of that larger physical/geographical entity.

Any TopologicalEntity that uses any part of that larger physical/geographical unit will suffer impact and hence each TopologicalEntity shares risk.

The identifier of each physical/geographical unit that is involved in the realization of each segment of a Topological entity can be listed in the RiskParameter_Pac of that TopologicalEntity.

A segment has one or more risk characteristic.

Shared risk between two TopologicalEntities compromises the integrity of any solution that use one of those TopologicalEntity as a backup for the other.

Where two TopologicalEntities have a common risk characteristic they have an elevated probability of failing simultaneously compared to two TopologicalEntities that do not share risk characteristics.

- riskCharacteristicList: A list of risk characteristics (RiskCharacteristic) for consideration in an analysis of shared risk. Each element of the list represents a specific risk consideration.

- RiskCharacteristic: The information for a particular risk characteristic where there is a list of risk identifiers related to that characteristic. It includes:

  o riskCharacteristicName: The name of the risk characteristic. The characteristic may be related to a specific degree of closeness. For example a particular characteristic may apply to failures that are localized (e.g. to one side of a road) where as another characteristic may relate to failures that have a broader impact (e.g. both sides of a road that crosses a bridge). Depending upon the importance of the traffic being routed different risk characteristics will be evaluated.

  o riskIdentifierList: A list of the identifiers of each physical/geographic unit (with the specific risk characteristic) that is related to a segment of the TopologicalEntity.

5.2.2. TransferCost_Pac

The cost characteristics of a TopologicalEntity not necessarily correlated to the cost of the underlying physical realization.

They may be quite specific to the individual TopologicalEntity e.g. opportunity cost. Relates to layer capacity

There may be many perspectives from which cost may be considered for a particular TopologicalEntity and hence many specifc costs and potentially cost algorithms.

Using an entity will incur a cost.

- costCharcteristicList: The list of costs (CostCharacteristic) where each cost relates to some aspect of the Link

- CostCharcteristic: The information for a particular cost characteristic

  - costName: The cost characteristic will related to some aspect of the TopologicalEntity (e.g. $ cost, routing weight). This aspect will be conveyed by the costName

  - costValue: The specific cost.

  - costAlgorithm: The cost may vary based upon some properties of the TopologicalEntity. The rules for the variation are conveyed by the costAlgorithm.

5.2.3. TransferTiming_Pac

A link will suffer effects from the underlying physical realization related to the timing of the information passed by the link.

- fixedLatencyCharacteristic: A TopologicalEntity suffers delay caused by the realization of the servers (e.g. distance related; FEC encoding etc.) along with some client specific processing. This is the total average latency effect of the TopologicalEntity

- jitterCharacteristic: High frequency deviation from true periodicity of a signal and therefore a small high rate of change of transfer latency. Applies to TDM systems (and not packet).

- wanderCharacteristics: Low frequency deviation from true periodicity of a signal and therefore a small low rate of change of transfer latency. Applies to TDM systems (and not packet).

- queuingLatencyList: The effect on the latency of a queuing process. This only has significant effect for packet based systems and has a complex characteristic (QueuingLatency).

  - QueuingLatency: Provides information on latency characteristic for a particular stated trafficProperty.

5.2.4. TransferIntegrity_Pac

Transfer integrity characteristic covers expected (specified) error, loss and duplicaion signal content as well as any damage of any form to total link and to the client signals.

- errorCharacteristic: describes the degree to which the signal propagated can be errored. Applies to TDM systems as the errored signal will be propagated and not packet as errored packets will be discarded.

- lossCharacteristic: Describes the acceptable characteristic of lost packets where loss may result from discard due to errors or overflow. Applies to packet systems and not TDM (as for TDM errored signals are propagated unless grossly errored and overflow/underflow turns into timing slips).

- repeatDeliveryCharacteristic: Primarily applies to packet systems where a packet may be delivered more than once (in fault recovery for example). It can also apply to TDM where several frames may be received twice due to switching in a system with a large differential propagation delay.

- deliveryOrderCharacteristic: Describes the degree to which packets will be delivered out of sequence. Does not apply to TDM as the TDM protocols maintain strict order.

- unavailableTimeCharacteristic: Describes the duration for which there may be no valid signal propagated.

- serverIntegrityProcessCharacteristic: Describes the effect of any server integrity enhancement process on the characteristics of the TopologicalEntity.

5.2.5. TransferCapcity_Pac

The TopologicalEntity derives capacity from the underlying realization.

A TopologicalEntity may be an abstraction and virtualization of a subset of the underlying capability offered in a view or may be directly reflecting the underlying realization.

A TopologicalEntity may be directly used in the view or may be assigned to another view for use.

The clients supported by a multi-layer TopologicalEntity may interact such that the resources used by one client may impact those available to another. This is derived from the LTP spec details.

A TopologicalEntity represents the capacity available to user (client) along with client interaction and usage.

A TopologicalEntity may reflect one or more client protocols and one or more members for each profile.

- totalPotentialCapacity: A “best case” view of the capacity of the TopologicalEntity assuming that any shared capacity is available to be taken.

Note that this area is still under development to cover concepts such as:

- exclusiveCapacityList: The capacity allocated to this TopologicalEntity for its exclusive use

- sharedCapacityList: The capacity allocated to this TopologicalEntity that is not exclusively available as it is shared with others.

- assignedAsExclusiveCapacityList: The capacity assigned from this TopologicalEnity to another TopologicalEntity for its exclusive use

- assignedAsSharedCapacityList: The capacity assigned to one or more other TopologicalEntities for shared use where the interaction follows some stated algorithm.

- Capacity which includes:

- totalSize
- numberOfUsageInstances
- maximumUsageSize
- numberingRange

5.2.6. Validation_Pac

Validation covers the various adjacenct discovery and reachability verification protocols. Also may cover Information source and degree of integrity.

- validationMechanismList: Provides details of the specific validation mechanism(s) used to confirm the presence of an intended topologicalEntity.

5.2.7. LayerProtocolTransition_Pac

Relevant for a Link that is formed by abstracting one or more LTPs (in a stack) to focus on the flow and deemphasize the protocol transformation.

This abstraction is relevant when considering multi-layer routing.

The layer protocols of the LTP and the order of their application to the signal is still relevant and need to be accounted for. This is derived from the LTP spec details.

This Pac provides the relevant abstractions of the LTPs and provides the necessary association to the LTPs involved.

Links that included details in this Pac are often referred to as Transitional Links.

- transitionedLayerProtocolList: Provides the ordered structure of layer protocol transitions encapsulated in the TopologicalEntity. The ordering relates to the LinkEnd role.

6. Usage of the CNM Topology Subset regarding TE Topology DM

As discussed earlier, a data model (DM) may be derived from an IM. It is possible to leverage the CNM Topology Subset to assess the consistency and completeness of related YANG modules under development. Appendix A provides a simple example of such a derivation.

7. Security Considerations

This informational document is intended only to provide a description of an interface-protocol-neutral information model, and the security concerns are therefore out of the scope of this document.

8. IANA Considerations

This document includes no request to IANA.

9. Conclusions

The information model described in this draft, which is relevant to Network Topology [ONF TR-512], can be leveraged in assessing the consistency and completeness of related YANG modules under development.

10. References

10.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

10.2. Informative References

[I-D.betts] Betts, M., Davis, N., Lam, K., Zeuner, B., Mansfield, S. and P. Doolan, “Framework for Deriving Interface Data Schema from UML Information Models”, draft-betts-netmod-framework-data-schema-uml-02 (work in progress), March 2015

[I-D.mansfield] Mansfield, S., Zeuner, B., Davis, N., Yun, X., Tochio, Y., Lam, K. and E. Varma, "Guidelines for Translation of UML Information Model to YANG Data Model", draft-mansfield-netmod-uml-to-yang-01 (work in progress), July 2015

[ONF TR-512] ONF TR-512 "ONF-CIM Core Model base document 1.0" (https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/Core_Information_Model_V1.0.pdf) + Model 1.0 (https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/CoreInformationModel.V1.0.zip), March 2015

[G.7711] Recommendation ITU-T G.7711/Y.17022 "Generic protocol-neutral information model for transport resources", August 2015

[G.852.2] Recommendation ITU-T G.852.2 "Enterprise viewpoint description of transport network resource model", March 1999

[TMF612] TM Forum 612 "MTOSI Information Agreement", October 2014

11. Acknowledgments

This document was prepared using 2-Word-v2.0.template.dot.

Appendix A YANG from the CNM Topology Subset

Shown below is the YANG specification for the object classes in the Topology subset of the CNM. To also illustrate the concept of pruning (see [I-D.betts]), not all of the attributes of the object classes (see Section 5.2) defined in the ONF-CIM CNM are taken for mapping to YANG.

The YANG module has been created using the simple mapping rules listed below, as described in [I-D.mansfield].

UML artifact	**YANG artifact**
identifiable object class	list statement
attribute	leaf statement
attribute list	leaf-list statement
non-identifiable object class (Pac)	container statement
attribute referring to data type	container statement
data type	grouping statement
attribute multiplicity	min/max-elements substatements

A.1. ONF-CIM CNM Topology YANG Specification

```
<CODE BEGINS> file "ONF_TR-512_CoreModel_Topolgy.yang"

// Contents of "Topology IM Draft for IETF-94"
   module ietf-TopologyIM {
      namespace "urn:ietf:params:xml:ns:yang:ietf-TopologyIM";
      // replace with iana namespace when assigned
      prefix "TopIM";
      organization "IETF";
      contact
         "WG Web:  <http://tools.ietf.org/wg/teas/>
         WG List: <mailto:teas@ietf.org>

         WG Chair: Vishnu Pavan Beeram
               <mailto:vbeeram@juniper.net>
```

```
             WG Chair: Lou Berger
                   <mailto:lberger@labn.net>

             Editor: Kam Lam
                 <mailto:kam.lam@alcatel-lucent.com>";

          description
             "Brief YANG example for Link object class in the Core
              Model (ONF::TR-512).";
          revision 2015-07-02 {
             description
                "Initial version";
          }

          grouping LayerProtocol {
            description
                "Each transport layer is represented by a LayerProtocol
                (LP) instance. The LayerProtocol instances it can be
                used for controlling termination and monitoring
                functionality.  It can also be used for controlling the
                adaptation (i.e. encapsulation and/or multiplexing of
                client signal), tandem connection monitoring, traffic
                conditioning and/or shaping functionality at an
                intermediate point along a connection. ";

            leaf guid {
                type string;
             }

             leaf layerProtocolName {
                type string;
             }

             leaf lpSpec {
                type string;
             }

             leaf configuredClientCapacity {
                type string;
             }

             leaf lpDirection {
                type string;
             }

             leaf terminationState {
                type string;
```

```
           }

        }

        list LayerProtocol {
          key "guid";

           uses LayerProtocol;
        }

        list ForwardingDomain {
          description
              "The ForwardingDomain (FD) object class models the
              topological component which represents the opportunity
              to enable forwarding (of specific transport
              characteristic information at one or more protocol
              layers) between points represented by the LTP in the
              model. ";
           key "guid";
           leaf guid {
              type string;
           }
           leaf-list layerProtocolNameList {
              type string;
              min-elements "1";
           }
           leaf-list lowerLevelFdRefList {
              type leafref {
                 path "/ForwardingDomain/guid";
              }
           }
           leaf-list fcRefList {
              type leafref {
                 path "/ForwardingConstruct/guid";
              }
           }

           leaf-list ltpRefList {
              type leafref {
                 path "/LogicalTerminationPoint/guid";
              }
           }

           leaf-list linkRefList {
              type leafref {
                 path "/Link/guid";
              }
```

```
         }

      }

      list LogicalTerminationPoint {
         description
            "The LogicalTerminationPoint (LTP) object class
            encapsulates the termination and adaptation functions
            of one or more transport layers.  The structure of LTP
            supports all transport protocols including circuit and
            packet forms.";

         key "guid";
         leaf guid {
            type string;
         }

         leaf-list serverLtpRefList {
            type leafref {
                path "/LogicalTerminationPoint/guid";
            }
         }

         leaf-list clientLtpRefList {
            type leafref {
                path "/LogicalTerminationPoint/guid";
            }
         }

         list lpList { // Should be leaf-list? (only refs?)
            key "guid";
            uses LayerProtocol;
         }

         leaf-list connectedLtpRef {
            type leafref {
                path "/LogicalTerminationPoint/guid";
            }
         }

         leaf-list peerLtpRef {
            type leafref {
                path "/LogicalTerminationPoint/guid";
            }
         }

         leaf ltpSpec {
            type string;
         }

         leaf physicalPortReference {
            type string;
         }

         leaf-list ltpRefList {
            type leafref {
                path "/LogicalTerminationPoint/guid";
            }
         }

         leaf ltpDirection {
            type string;
         }
      }

      list ForwardingConstruct {
          description
            "The ForwardingConstruct (FC) object class models enabled
            potential for forwarding between two or more LTPs at a
            particular specific layerProtocol.
            Like the LTP the FC supports any transport protocol
            including all circuit and packet forms. It is used to
            effect forwarding of transport characteristic (layer
            protocol) information. An FC can be in only one FD.
            The ForwardingConstruct is a Forwarding entity.";
         key "guid";
         leaf guid {
            type string;
         }

         leaf-list lowerLevelFcRefList {
             type leafref {
               path "/ForwardingConstruct/guid";
            }
         }

         leaf-list fcRouteRefList {
             type leafref {
               path "/ForwardingConstruct/guid";
            }
         }

         list endPointList {
             key "guid";
             leaf guid {
                type string;
             }
             leaf-list ltpRefList {
                type leafref {
                    path "/LogicalTerminationPoint/guid";
                }
                max-elements "2";
             }
             leaf role {
                type string;
             }
             leaf endPointDirection {
                type string;
             }

         }

         list fcSwitchList {
             description
                "The FcSwitch object class models the switched
                forwarding of traffic (traffic flow) between Eps
                and is present where there is protection
                functionality in the FC.  If an FC exposes
                protection (having two EndPoints that provide
                alternative identical inputs/outputs), the FC
                will have one or more associated FcSwitch objects
                to represent the alternative flow choices visible
                at the edge of the FC. The FC switch represents
                and defines a protection switch structure
                encapsulated in the FC.  ";

             key "guid";
             leaf guid {
                type string;
             }

             leaf holdOffTime {
                type uint64;
             }
             leaf waitToRestoreTime {
                type uint64;
             }
             leaf protType {
                type string;
             }
             leaf operType {
                type string;
             }
             leaf-list selectedEndPointRefList {
                type leafref {
                    path "/ForwardingConstruct/endPointList/guid";
                }
             }
             /**leaf-list profileProxyRefList {
                type leafref {
                    path ""; // To be defined
                }
             }*/
             /**leaf-list configurationAndSwitchControlRef {
                type leafref {
                    path ""; // To be defined
                }
             }*/

         }

         leaf forwardingDirection {
            type string;
         }
      }

      list Link {
         key "guid";
         leaf guid {
            type string;
         }
         uses TopologicalEntity;
         leaf-list layerProtocolNameList {
            type string;
            min-elements "1";
         }
         leaf-list aggregatedLinkList {
            type leafref {
               path "/Link/guid";
            // Mapping of recursion still to be agreed
            }
         }

         list LinkEnd {
            key "localId";
            leaf localId {
               type string;
            }
```

```
                min-elements "2";
                leaf role {
                   type string; // Undefined in the model
                   mandatory true;
                }
                leaf-list localIdList {
                   type string; // YANG data type for name-value-pair?
                   min-elements "1";
                }
                leaf offNetworkAddress {

                   type string;
                }
                leaf-list ltpRefList {
                   type leafref {
                      path "/LogicalTerminationPoint/guid";
                   }
                }
             }
          }

       // Groupings

             grouping TopologicalEntity {

             // Conditional Pacs associated to TopologicalEntity

             container RiskParameter_Pac {
                presence
                   "<condition for RiskParameter_Pac attributes>";
                list riskCharacteristicList {
                   key "riskCharacteristicName";
                   leaf riskCharacteristicName {
                      type string;
                   }
                   leaf-list riskIdentifierList {
                      type string;
                      min-elements "1";
                   }
                   min-elements "1";
                }
             }

             container TransferCost_Pac {
                presence
                   "<condition for TransferCost_Pac attributes>";
                list costCharacteristicList {
                   key "costName";
                   leaf costName {
                      type string;
                   }
                   leaf costValue {
                      type string;
                      mandatory true;
                   }
                   leaf costAlgorithm {
                      type string; // Undefined in the model
                   }
                   min-elements "1";

                }
             }

             container TransferTiming_Pac {
                presence
                   "<condition for TransferTiming_Pac attributes>";
                leaf fixedLatencyCharacteristic {
                   type string;
                   mandatory true;
                }
                leaf jitterCharacteristic {
                   type string;
                }
                leaf wanderCharacteristic {
                   type string;
                }
                list queuingLatencyList {
                   key "trafficProperty";
                   leaf trafficProperty {
                      type string;
                   }
                   leaf latencyForTrafficWithProperty {
                      type string;
                      mandatory true;
                   }
                }
             }

             container TransferIntegrity_Pac {
                presence
                   "<condition for TransferIntegrity_Pac attributes>";
                leaf errorCharacteristic {
                   type string;
                }
```

```
                leaf lossCharacteristic {
                   type string;
                }
                leaf repeatDeliveryCharacteristic {
                   type string;
                }
                leaf deliveryOrderCharacteristic {
                   type string;
                }
                leaf unavailableTimeCharacteristic {
                      type string;
                      mandatory true;
                }
                leaf serverIntegrityProcessCharacteristic {
                   type string;

                }
             }

             container TransferCapacity_Pac {
                presence
                   "<condition for TransferCapacity_Pac attributes>";
                container totalPotentialCapacity {
                   uses Capacity;
                }
                container availableCapacity {
                   uses Capacity;
                }
                container capacityAssignedToUserView {
                   // Don't know how to make this attribute multi-valued
                   uses Capacity;
                }
                leaf capacityInteractionAlgorithm {
                   type string;
                }
             }

             container Validation_Pac {
                presence "<condition for Validation_Pac attributes>";
                list validationMechanismList {
                   key "validationMechanism";
                   leaf validationMechanism {
                      type string;
                   }
                   leaf layerProtocolAdjacencyValidated {
                      type string;
                      mandatory true;
```

```
              }
              leaf validationRobustness {
                 type string;
                 mandatory true;
              }
              min-elements "1";
           }
        }

        container LayerProtocolTransition_Pac {
           // Don't know how to make this Pac multi-valued
           presence "<condition for LayerProtocolTransition_Pac
                   attributes>";
           leaf-list transitionedLayerProtocolList {
              type string;
              min-elements "1";
           }

        }
     }

     grouping Capacity {
        leaf totalSize {
           type string;
           mandatory true;
        }
        leaf numberOfClientInstances {
           type string;
        }
        leaf maximumClientSize {
           type string;
        }
        leaf numberingRange {
           type string;
        }
     }
 } <CODE ENDS>
```

A.2. Tree-Style Summary of Topology YANG Specification

```
   module: ietf-TopologyIM
      +--rw LayerProtocol* [guid]
      |  +--rw guid                        string
      |  +--rw layerProtocolName?          string
      |  +--rw lpSpec?                     string
      |  +--rw configuredClientCapacity?   string
      |  +--rw lpDirection?                string
      |  +--rw terminationState?           string
      +--rw ForwardingDomain* [guid]
      |  +--rw guid                     string
      |  +--rw layerProtocolNameList*   string
      |  +--rw lowerLevelFdRefList*     -> /ForwardingDomain/guid
      |  +--rw fcRefList*               -> /ForwardingConstruct/guid
      |  +--rw ltpRefList*              -> /LogicalTerminationPoint/guid
      |  +--rw linkRefList*             -> /Link/guid
      +--rw LogicalTerminationPoint* [guid]
      |  +--rw guid                     string
      |  +--rw serverLtpRefList*        -> /LogicalTerminationPoint/guid
      |  +--rw clientLtpRefList*        -> /LogicalTerminationPoint/guid
      |  +--rw lpList* [guid]
      |  |  +--rw guid                        string
      |  |  +--rw layerProtocolName?          string
      |  |  +--rw lpSpec?                     string
      |  |  +--rw configuredClientCapacity?   string
      |  |  +--rw lpDirection?                string
      |  |  +--rw terminationState?           string
      |  +--rw connectedLtpRef*         -> /LogicalTerminationPoint/guid
      |  +--rw peerLtpRef*              -> /LogicalTerminationPoint/guid
      |  +--rw ltpSpec?                 string
      |  +--rw physicalPortReference?   string
      |  +--rw ltpRefList*              -> /LogicalTerminationPoint/guid
      |  +--rw ltpDirection?            string
      +--rw ForwardingConstruct* [guid]
      |  +--rw guid                   string
      |  +--rw lowerLevelFcRefList*   -> /ForwardingConstruct/guid
      |  +--rw fcRouteRefList*        -> /ForwardingConstruct/guid
      |  +--rw endPointList* [guid]
      |  |  +--rw guid                 string
      |  |  +--rw ltpRefList*          -> /LogicalTerminationPoint/guid
      |  |  +--rw role?                string
      |  |  +--rw endPointDirection?   string
      |  +--rw fcSwitchList* [guid]
      |  |  +--rw guid                        string
      |  |  +--rw holdOffTime?                uint64
      |  |  +--rw waitToRestoreTime?          uint64
      |  |  +--rw protType?                   string
      |  |  +--rw operType?                   string
      |  |  +--rw selectedEndPointRefList*   ->
   /ForwardingConstruct/endPointList/guid
      |  +--rw forwardingDirection?   string
      +--rw Link* [guid]
         +--rw guid                           string
         +--rw RiskParameter_Pac!
         |  +--rw riskCharacteristicList* [riskCharacteristicName]
         |     +--rw riskCharacteristicName    string
         |     +--rw riskIdentifierList*       string
         +--rw TransferCost_Pac!
         |  +--rw costCharacteristicList* [costName]
         |     +--rw costName         string
         |     +--rw costValue        string
         |     +--rw costAlgorithm?   string
         +--rw TransferTiming_Pac!
         |  +--rw fixedLatencyCharacteristic    string
         |  +--rw jitterCharacteristic?         string
         |  +--rw wanderCharacteristic?         string
         |  +--rw queuingLatencyList* [trafficProperty]
         |     +--rw trafficProperty                  string
         |     +--rw latencyForTrafficWithProperty    string
         +--rw TransferIntegrity_Pac!
         |  +--rw errorCharacteristic?                    string
         |  +--rw lossCharacteristic?                     string
         |  +--rw repeatDeliveryCharacteristic?           string
         |  +--rw deliveryOrderCharacteristic?            string
         |  +--rw unavailableTimeCharacteristic           string
         |  +--rw serverIntegrityProcessCharacteristic?   string
         +--rw TransferCapacity_Pac!
         |  +--rw totalPotentialCapacity
         |  |  +--rw totalSize                 string
         |  |  +--rw numberOfClientInstances?   string
         |  |  +--rw maximumClientSize?         string
         |  |  +--rw numberingRange?            string
         |  +--rw availableCapacity
         |  |  +--rw totalSize                 string
         |  |  +--rw numberOfClientInstances?   string
         |  |  +--rw maximumClientSize?         string
         |  |  +--rw numberingRange?            string
         |  +--rw capacityAssignedToUserView
         |  |  +--rw totalSize                 string
         |  |  +--rw numberOfClientInstances?   string
         |  |  +--rw maximumClientSize?         string
         |  |  +--rw numberingRange?            string
         |  +--rw capacityInteractionAlgorithm?   string
         +--rw Validation_Pac!
         |  +--rw validationMechanismList* [validationMechanism]
         |     +--rw validationMechanism               string
         |     +--rw layerProtocolAdjacencyValidated    string
         |     +--rw validationRobustness               string
         +--rw LayerProtocolTransition_Pac!
         |  +--rw transitionedLayerProtocolList*   string
         +--rw layerProtocolNameList*         string
         +--rw aggregatedLinkList*            -> /Link/guid
```

```
        +--rw LinkEnd* [localId]
           +--rw localId             string
           +--rw role                string
           +--rw localIdList*        string
           +--rw offNetworkAddress?  string
           +--rw ltpRefList*         -> /LogicalTerminationPoint/guid
```

Appendix B YANG for Transport API Topology Service

A YANG module for a Transport API Topology Service is illustrated in Section B.2. The YANG module is based on the Transport API Topology information model described in Section B.1, which has been derived from the ONF-CIM CNM Topology model through pruning and refactoring targeted to support this purpose specific view. As in Appendix A, the YANG module has been created using the simple mapping rules described in [I-D.mansfield].

B.1. Transport API Topology Service Information Model

The T-API Topology Service IM is a purpose-specific IM, derived from the ONF-CIM CNM Topology model following the principles described in [I-D.betts]. It uses terminology that is familiar to the transport network management industry and maps to the constructs defined in the ONF-CIM CNM Topology model.

The API context is the scope and control and naming that a particular T-API Topology Service provider or its client application has with respect to the information exchanged over the interface.

B.1.1. T-API Topology Service – Topology Constructs

This section outlines the topology constructs utilized in the Transport API Topology Service IM, and their relationship to the topology subset of the CNM described within Sections 3 and 4.

o *Topolog*y: represents the transparent topological aspects of the ForwardingDomain (FD). Topology describes the underlying topological network of Nodes and Links that enable the forwarding function provided by the FD.

o *Node*: represents the opaque forwarding aspects of the ForwardingDomain (FD). Node describes the edge ports of the FD (*NodeEdgePoint*) and the forwarding capabilities between those edge ports.

o *Link*: represents the effective adjacency between two or more associated Nodes in a Topology (same as CNM). Link is terminated by NodeEdgePoints of the associated Nodes.

- *NodeEdgePoint* represents the edge port (LTP) functions for accessing the forwarding capabilities of the Node. It provides an encapsulation of addressing, mapping, termination, adaptation and OAM functions of one or more transport layers (including circuit and packet forms) performed at the entry/exit points of Node.

B.1.2. T-API Topology Service - Retrieval APIs

The T-API Topology Services API enables the API client to get:

- Topology details: returns attributes of the Topology identified by the provided input ID. This includes references to lower-level Nodes and Links encompassed by that Topology.

- Node details: returns attributes of the Node identified by the provided input ID. Includes references to NodeEdgePoints aggregated by the Node, and attributes representing the identification, naming, states and forwarding capabilities of the Node.

- Link details: returns attributes of the Link identified by the provided input ID. Includes references to NodeEdgePoints terminating the Link, and references to the Nodes associated by the Link.

- NodeEdgePoint details: returns attributes of the NodeEdgePoint identified by the provided input ID.

The API supports a retrieve-scope filter: LayerProtocol list. If set, the API call will return output that is relevant to the specified Layer only.

B.2. Transport API Topology Service YANG Specification

```
<CODE BEGINS> file "ONF_TAPI_Topolgy.yang"

// Contents of "Topology Service Draft for IETF-94"
```

```
   module TapiTopologyService {

     namespace "urn:onf:Tapi:TopologyService:yang";
     prefix TapiTopologyService;

     organization "ONF T-API Group";

     contact "mailto:info@opennetworking.org";

     description
       "This module contains a collection of YANG definitions for
        Topology Service. It is part of the T-API
        specification.
        Authors List:
        Ricard Vilalta (CTTC)
        Victor López (Telefnica)
        (T-API Team)";

     revision 2015-09-11 {
       description
         "First draft for Transport API Topology Service";
     }

     /*****************************************************************
     * Groupings
     *****************************************************************/

     grouping LayerProtocol {
       leaf layerProtocol {
         type string;
       }
     }

     grouping LayerProtocolName {
       leaf layerProtocolName {
         type string;
       }
     }

     grouping EndLtpRole {
       leaf endLtpRole {
         type string;
       }
     }

     grouping NodeEdgePoint {
       description
       "The LogicalTerminationPoint (LTP) object class encapsulates the
       termination and adaptation functions of one or more transport
       layers. The structure of LTP supports all transport protocols
       including circuit and packet forms.";

       leaf nodeEdgePointId {
         type string;
       }

       list ipList{
         key "layerProtocol";
         uses LayerProtocol;
       }
     }

     grouping Node {
       description
         "The ForwardingDomain (FD) object class models the
   ForwardingDomain topological component which is used to effect
   forwarding of transport characteristic information and offers
   the potential to enable forwarding.
         At the lowest level of recursion, an FD (within a network
   element (NE)) represents a switch matrix (i.e., a fabric). Note
   that an NE can encompass multiple switch matrices (FDs). "

       leaf nodeId {
         type string;
       }

       leaf-list _encapTopologyRef {
         type leafref {
           path "/Topology/topologyId";
         }
         max-elements "1";
       }

       list edgeLtpList { // Differs from UML
         key "nodeEdgePointId";
         uses NodeEdgePoint;
       }

       list layerProtocolNameList {
         key "layerProtocolName";
         uses LayerProtocolName;
         min-elements "1";
       }
     }
     grouping Link {
       description
       "The Link object class models effective adjacency between two or
   more ForwardingDomains (FD).";

       leaf linkId {
         type string;
       }

       leaf-list _nodeRefList{
         type leafref {
           path "/Topology/node/nodeId";
         }
       }

       leaf-list _endLtpRefList {
         type leafref {
           path "/Topology/node/edgeLtpList/nodeEdgePointId";
         }
         min-elements "2";
       }

       list endLtpRoleList {
         key "endLtpRole";
         uses EndLtpRole;
         min-elements "2";
       }

       list layerProtocolNameList {
         key "layerProtocolName";
         uses LayerProtocolName;
         min-elements "1";
       }
     }

     grouping Topology {
       leaf topologyId {
         type string;
       }

       list node {
         key "nodeId";
         uses Node;
       }

       list link {
         key "linkId";
         uses Link;
       }

       uses LayerProtocolName;
     }



     /************************************************************
     * Main containers
     ************************************************************/


     list Topology {
       description
       "The ForwardingConstruct (FC) object class models enabled
   potential for forwarding between two or more LTPs and like the
   LTP supports any transport protocol including all circuit and
   packet forms.
       At the lowest level of recursion, a FC represents a
   cross-connection within an NE.";

       key "topologyId";
       uses Topology;
     }



     /************************************************************
     * RPC definitions
     ************************************************************/

     rpc getTopologyDetails {
       input {
         leaf topologyId {
           description "topology id";
           type string;
         }
       }

       output {
         container topology {
             uses Topology;
         }
       }
     }
```

```
     rpc getLinkDetails {
       input {
         leaf linkId {
           description "link id";
           type string;
           }
       }
       output {
         container link{
           uses Link;
         }
       }
     }

     rpc getNodeDetails {
       input {
         leaf nodeId {
           description "node id";
           type string;
           }
       }
       output {
         container node{
           uses Node;
         }
       }
     }

     rpc getNodeEdgePointDetails {
       input {
         leaf nodeEdgePointId {
           description "node edge point id";
           type string;
           }
       }
       output {
         container nodeEdgePoint{
           uses NodeEdgePoint;
         }
       }
     }
    } <CODE ENDS>
```

B.3. Tree-Style Summary of Topology Service YANG Specification

```
   module: TapiTopologyService
      +--rw Topology* [topologyId]
         +--rw topologyId           string
         +--rw node* [nodeId]
         |  +--rw nodeId                   string
         |  +--rw _encapTopologyRef*       -> /Topology/topologyId
         |  +--rw edgeLtpList* [nodeEdgePointId]
         |  |  +--rw nodeEdgePointId    string
         |  |  +--rw ipList* [layerProtocol]
         |  |     +--rw layerProtocol    string
         |  +--rw layerProtocolNameList* [layerProtocolName]
         |     +--rw layerProtocolName    string
         +--rw link* [linkId]
         |  +--rw linkId                   string
         |  +--rw _nodeRefList*            -> /Topology/node/nodeId
         |  +--rw _endLtpRefList*          ->
   /Topology/node/edgeLtpList/nodeEdgePointId
         |  +--rw endLtpRoleList* [endLtpRole]
         |  |  +--rw endLtpRole    string
         |  +--rw layerProtocolNameList* [layerProtocolName]
         |     +--rw layerProtocolName    string
         +--rw layerProtocolName?   string
   rpcs:
      +---x getTopologyDetails
      |  +---w input
      |  |  +---w topologyId?   string
      |  +--ro output
      |     +--ro topology
      |        +--ro topologyId?          string
      |        +--ro node* [nodeId]
      |        |  +--ro nodeId                   string
      |        |  +--ro _encapTopologyRef*       -> /Topology/topologyId
      |        |  +--ro edgeLtpList* [nodeEdgePointId]
      |        |  |  +--ro nodeEdgePointId    string
      |        |  |  +--ro ipList* [layerProtocol]
      |        |  |     +--ro layerProtocol    string
      |        |  +--ro layerProtocolNameList* [layerProtocolName]
      |        |     +--ro layerProtocolName    string
      |        +--ro link* [linkId]
      |        |  +--ro linkId                   string
      |        |  +--ro _nodeRefList*            -> /Topology/node/nodeId
      |        |  +--ro _endLtpRefList*          ->
   /Topology/node/edgeLtpList/nodeEdgePointId
      |        |  +--ro endLtpRoleList* [endLtpRole]
      |        |  |  +--ro endLtpRole    string
      |        |  +--ro layerProtocolNameList* [layerProtocolName]
      |        |     +--ro layerProtocolName    string
      |        +--ro layerProtocolName?   string
      +---x getLinkDetails
      |  +---w input
      |  |  +---w linkId?   string
      |  +--ro output
      |     +--ro link
      |        +--ro linkId?                  string
      |        +--ro _nodeRefList*            -> /Topology/node/nodeId
      |        +--ro _endLtpRefList*          ->
   /Topology/node/edgeLtpList/nodeEdgePointId
      |        +--ro endLtpRoleList* [endLtpRole]
      |        |  +--ro endLtpRole    string
      |        +--ro layerProtocolNameList* [layerProtocolName]
      |           +--ro layerProtocolName    string
      +---x getNodeDetails
      |  +---w input
      |  |  +---w nodeId?   string
      |  +--ro output
      |     +--ro node
      |        +--ro nodeId?                  string
      |        +--ro _encapTopologyRef*       -> /Topology/topologyId
      |        +--ro edgeLtpList* [nodeEdgePointId]
      |        |  +--ro nodeEdgePointId    string
      |        |  +--ro ipList* [layerProtocol]
      |        |     +--ro layerProtocol    string
      |        +--ro layerProtocolNameList* [layerProtocolName]
      |           +--ro layerProtocolName    string
      +---x getNodeEdgePointDetails
         +---w input
         |  +---w nodeEdgePointId?   string
         +--ro output
            +--ro nodeEdgePoint
               +--ro nodeEdgePointId?   string
               +--ro ipList* [layerProtocol]
                  +--ro layerProtocol    string
```

Authors' Addresses

Kam Lam
Alcatel-Lucent, USA

Phone: +1 732 331 3476
Email: kam.lam@alcatel-lucent.com

Eve Varma
Alcatel-Lucent, USA

Email: eve.varma@alcatel-lucent.com

Paul Doolan
Coriant, Germany

Phone: +1 972 357 5822
Email: paul.doolan@coriant.com

Malcolm Betts
ZTE, China

Phone: +1 678 534 2542
Email: malcolm.betts@zte.com.cn

Nigel Davis
Ciena, UK

Email: ndavis@ciena.com

Bernd Zeuner
Deutsche Telekom,   Germany

Phone: +49 6151 58 12086
Email: b.zeuner@telekom.de

Italo Busi
Huawei, China

Email: Italo.Busi@huawei.com

Scott Mansfield
Ericsson, Sweden

Phone: 1 724 931 9316
Email: scott.mansfield@ericsson.com

Yuji Tochio
Fujitsu, Japan

Phone: 81 44 754 8829
Email: tochio@jp.fujitsu.com

Ricard Vilalta
CTTC, Spain

Phone:
Email: ricard.vilalta@cttc.es

Victor Lopez
Telefonica, Spain

Phone:
Email: victor.lopezalvarez@telefonica.com